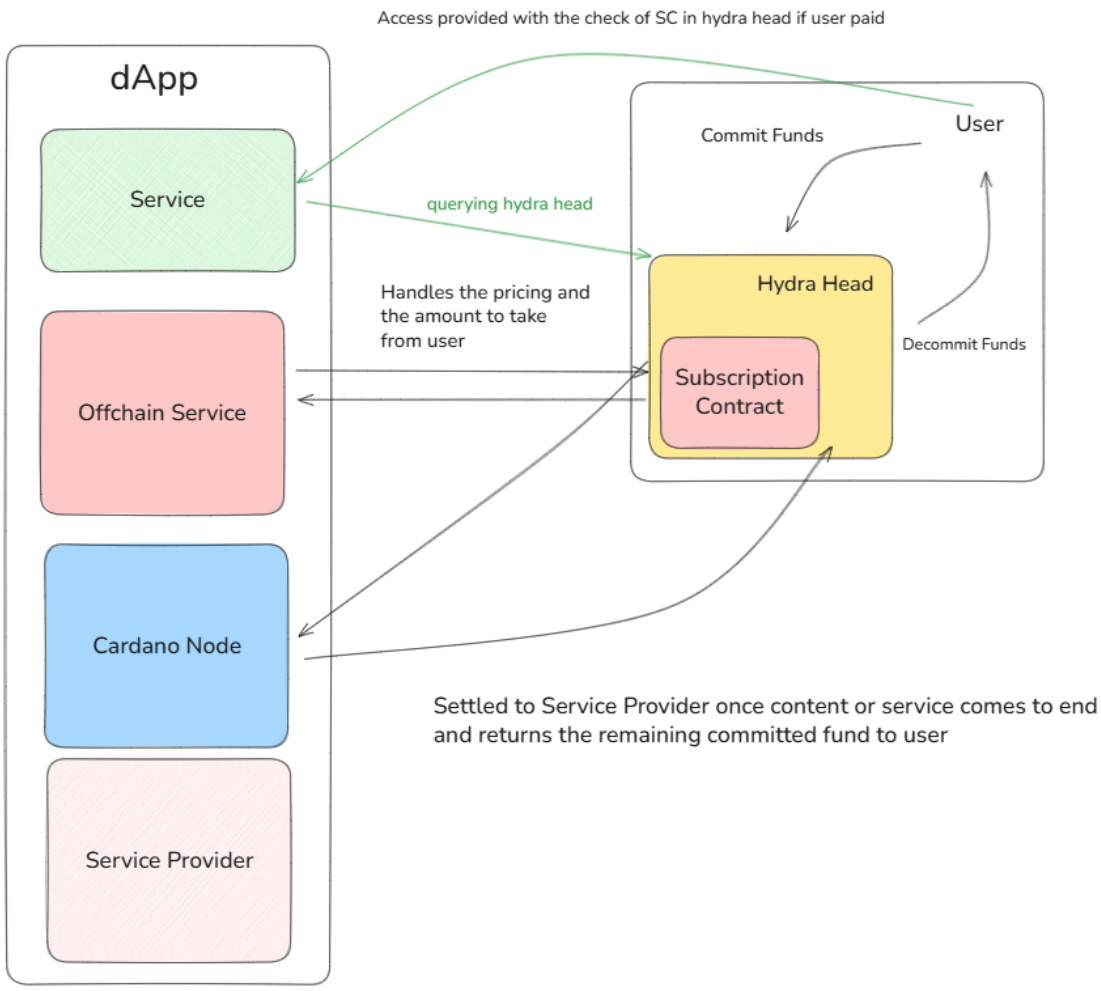
Hydra Subscription and Pay-as-You-Go Library on Cardano - Technical Feasibility report

Introduction

The Subscription and Pay-as-You-Go (PAYG) Library enables decentralized access to digital content through recurring payments or per-use billing on the Cardano blockchain. It integrates Hydra Heads to support scalable, high-throughput microtransactions and uses Aiken smart contracts for trustless subscription enforcement.

This document outlines the architectural design, smart contract logic, Hydra deployment, API structure, and testing strategy to realize this system.

High-Level Overview



This library enables content platforms to offer:

- Subscriptions with ADA or custom tokens.
- PAYG access validated by smart contracts.
- Hydra-powered microtransactions for performance.
- Trustless access verification via inline datums.

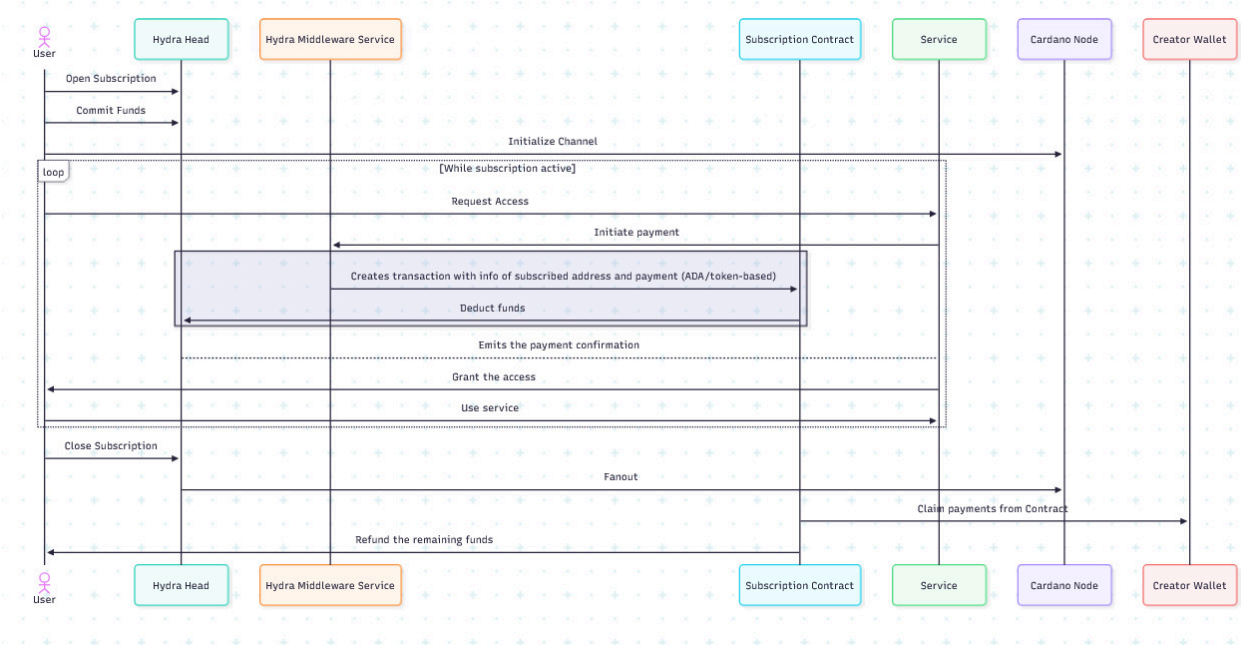
Payment Options

- **ADA-Based Subscriptions**: Use Cardano's native currency.
- **Token-Based Subscriptions**: Accept platform-issued or third-party tokens.

Hydra at a Glance

- Off-chain protocol for fast transaction batching.
- Finality achieved via L1 commitment.
- Suitable for service microtransactions.

Hydra PAYG and Subscriptions Library Detailed Architecture



System Components

- **Aiken Validator**: Governs subscription logic.
- **Hydra Nodes**: Enable fast PAYG flows.
- **Access Gateway**: Grants access post-payment verification.
- **Wallet Interface**: Initiates and signs user transactions.
- **Backend API**: Links smart contract output to service authorization.

Subscription Flow

1. User selects plan.

2. Submits transaction with correct `SubDatum` and `SubRedeemer`.
3. Platform co-signs.
4. Validator checks payment value, time interval, and datum consistency.
5. If valid, content gateway grants access.

Smart Contract Design

The core validator `subs` governs subscription transactions.

Key Types

```
SubRedeemer {
  PayADA(Subscription)
  PayToken(Subscription)
  Update
}

Subscription {
  subscribedAddress: PubKeyHash,
  adaValue: Int,
  lastPayment: PosixTime,
  nextPayment: PosixTime,
  intervalBetweenPayments: Int,
  policy_id: PolicyId,
  asset_name: AssetName,
  token_value: Int,
}

SubDatum {
  subscriber: PubKeyHash,
  platformAddress: PubKeyHash,
  subscription: Option<Subscription>,
}
```

Smart Contract Inputs and Outputs

Inputs:

- The current state of the subscription including subscriber address, platform address, and optional subscription record.
- Indicates the action being taken (ADA payment, Token payment, or Update).
- Provides context about the transaction, including inputs, outputs, validity range, and extra signatories.

Outputs:

- Updated UTXO with new: Contains the updated subscription data including new `lastPayment` and `nextPayment`.
- **Value Transfer Output**: Sends ADA or tokens to the subscriber address as part of the access rights.
- **Content access trigger**: Not on-chain but derived from successful output and datum validation by backend services.

Core Logic Checks

- Ensures correct ADA sent, valid timing, and matching datums.
- Checks token amount instead of ADA.
- Allows subscriber to modify terms if signed correctly.

Security Features

- `signReq()` ensures platform/subscriber sign TX.
- Output and datum comparisons validate state continuity.
- Interval between `lastPayment` and `nextPayment` enforced.

Hydra-Integrated PAYG and Subscriptions - How It Works

1. Spin-Up & Fund the Head

- A user and the service provider open a **single-party (or two-party) Hydra Head**.
- The user **commits one or more UTXOs** (e.g., 20 ADA) that become the spending balance for the session.

2. Real-Time Usage Inside Hydra

- Each content request or API call is an **instant L2 transaction**—milliseconds, near-zero fee.
- The same validator logic you would write for on-chain scripts runs **off-chain**, guaranteeing enforcement without block-chain latency.

3. Graceful Exit & Settlement

- When the user ends the session (or the balance is exhausted) either party issues `Close`.
- After the short contestation window, `Fanout` automatically returns unused funds to the user and routes earned revenue to the provider—**all on layer 1**.

Perfect Fits for This Pattern

Vertical	Why Hydra shines
Video / music streaming	Bill per second or per segment with sub-second authorisation.
Pay-per-view & live events	Collect once, stream instantly, settle when the show ends.
Time-metered APIs & SaaS	Charge per request or per CPU-second without clogging L1.

Hydra-PAYG and Subscriptions Library APIs

Top-level helper functions

Function	Purpose	I/O summary
`prepareCommit(fundingAddr, amount)`	1. Generates a fresh **Cardano key-pair** and **Hydra key-pair**.2. Builds an *unsigned* **Commit transaction** that locks `amount` into an upcoming single-party head.3. Returns everything the wallet needs to sign & submit.	**Input**• `fundingAddr` – the user's L1 address that holds ADA / tokens.• `amount` – `{ unit: "lovelace"
`openHead(keys)`	Uses the signed Commit Tx + keys to **start the Hydra Head** (single participant). Waits until the head is fully *open* and returns its public WebSocket / REST URL.	**Input** – `{ cardanoSk, hydraSk, headSeed }` (from previous step)Optional `{ contestationPeriodSeconds }` .**Output** – `{ headId: string, apiUrl: string }`

`Subscription` class

Represents **one user's live subscription channel** (i.e., the open Hydra Head and the embedded Subscription Contract script).

Constructor

```
ts
CopyEdit
new Subscription(headInfo: {
  apiUrl: string;
  headId: string;
  hydraSk: string;      // to sign L2 txs
}, params: {
  creatorAddress: string;         // where revenue will go
  validDuration: number;          // seconds
  payment: "ADA" | { unit: string };
  type: "fixed" | "dynamic";
```

```
});
```

Creates the helper object, starts listening to head events, and pre-computes the funding address derived from `cardanoVk`.

Static factory

`Subscription.from(url, keys)`

Re-hydrates a `Subscription` after page reload / server restart.

Methods

Method	What it does	Returns / rejects
`.subscribe(quantity?)`	Builds and pushes an off-chain payment to the Subscription Contract. If the plan is `type:"fixed"` the amount is taken from constructor; otherwise caller can pass a `quantity` each time.	`{ txHash, ticketId, snapshot }` (for auditing). Rejects with `"INSUFFICIENT_BALANCE"` or `"TX_INVALID"`.
`.getSubscription(txId)`	Fetches contract datum + status for a given payment. Useful for dashboards.	`{ ticketId, paidAt, amount, confirmations }`
`.closeSub(refundAddr?)`	Sends `Close` to the head, waits for `Fanout`, then resolves when the refund UTxO hits `refundAddr` (default = funding key's Addr).	`{ fanoutTxHash, refundAmount }`
`.claimSub(signer, contractAddr)`	After Fan-out, submits or signs the revenue-claim Tx that moves accumulated funds from `contractAddr` to the creator's wallet.	`{ claimTxHash }`
`.on(event, cb)`	Event emitter. Fires:• `"NewSub"` when a TxValid for this contract appears• `"UnSub"` when user closes•	returns `this` for chaining

Method	What it does	Returns / rejects
	`"FailedSub"` TxInvalid• `"InvalidSub"` datum mismatch / timeout	

Typical flow (pseudo-code)

```
ts
CopyEdit
// 1. prepare & fund
const { cbor, ...keys } = await prepareCommit(userAddr, {
  unit: "lovelace",
  quantity: 20_000_000n,
});
await walletApi.signAndSubmit(cbor);          // user signs L1 commit

// 2. open head
const { apiUrl, headId } = await openHead(keys);

// 3. create subscription helper
const sub = new Subscription(
  { apiUrl, headId, hydraSk: keys.hydraSk },
  {
    creatorAddress: CREATOR_ADDR,
    validDuration: 86_400,         // 1 day
    payment: "ADA",
    type: "dynamic",
  },
);

// 4. listen
sub.on("NewSub", r ⇒ tokenGateway.grant(r.ticketId));
sub.on("UnSub", () ⇒ tokenGateway.revoke(userId));

// 5. every API call / video chunk
```

```
await sub.subscribe(1_000_000n);   // pay 1 ADA inside head

// 6. user cancels
await sub.closeSub(refundAddr);

// 7. creator claims revenue
await sub.claimSub(creatorWalletSigner, CONTRACT_ADDR);
```

High-Throughput & Trustlessness - Key Challenges & Mitigation

Risk	Impact	Mitigation
Fan-out size cap (≈80 outputs)	Final settlement can fail	Limit outputs/head; auto-rotate to new head before cap
Growing event log	Slow restart	Enable `--persistence-rotate-after`
Ticket replay	Double access	Store `ticketId` hash; reject duplicates
Wallet lag spikes	UX stalls	Pre-pay small ticket buffer; async retry on failure
Node downtime	Missed contestation window	Run 1+ Hydra nodes

Dependency Matrix

Zone	Dependency
L1	Cardano Blockchain
L2	Hydra
Smart Contracts	Aiken
Wallet	CIP-30 (Nami, Eternl, Lace)
Tx-Builder	Mesh, Lucid

Testing Plans

Functional Testing

- `PayADA` and `PayToken` with correct/incorrect values
- Interval mismatch rejection
- Unauthorized access attempts

Compatibility Testing

- Wallet integration across Nami, Eternl, Lace
- Hydra node message handling
- Aiken validator on L1 vs Hydra

Integration Testing

- Digital Content Portals (e.g., NuCast)
- Pay as You go model for tech infrastructure providers